

EL624352621

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Access Redirector and Entry Reflector

Inventors:

Barry Bond

ATM Shafiqul Khalid

ATTORNEY'S DOCKET NO. MS1-571US

1 **TECHNICAL FIELD**

2 This invention relates to interoperability and compatibility of differing
3 versions of program modules using a common configuration data structure.
4

5 **BACKGROUND**

6 For each version of an operating system (OS), there are corresponding
7 versions of applications. For example, a 16-bit application is designed to run on a
8 16-bit OS (such as Microsoft® Windows 3.1®). An example of another version is
9 a 32-bit application, which is designed to run on a 32-bit OS (such as Microsoft®
10 Windows® 98).
11

12 Specific versions of applications are designed to operate under a specific
13 version of a specific OS. Furthermore, that specific version of a specific OS is
14 designed to operate with specific computing hardware (such as a specific
15 microprocessor).
16

17 For example, Microsoft® Office 2000 is an application designed to operate
18 on 32-bit versions of operating systems from the Microsoft Corporation.
19 Examples of such 32-bit operating systems include Windows® 2000, Windows
20 NT® 4.0 (and earlier), Windows® 98, and Windows® 95. The 32-bit OSs are
21 designed to operate on 32-bit compatible processors using an 32-bit instruction set
(such as the Intel™ Pentium™, Pentium II™, and Pentium III™).
22

23 **“Version”**

24 Herein, a reference to a “version” of a program module (such as an
25 application) refers to a class of module that is designed to run under a level of

operating system that is different than a previous version of the same program module. A new release of a program model is considered a new "version" when it is more than an incremental improvement and change.

For example, Microsoft® Word 95 is a version of the application that is different from Microsoft® Word 6.0. This is because Microsoft® Word 95 is designed to run on 32-bit OSs (such as Microsoft® Windows® 95 or later version), but it cannot run on 16-bit OSs (such as Microsoft® Windows® 3.1). Conversely, Microsoft® Word 97 is not a different version from Microsoft® Word 95 because both are designed to work on 32-bit OSs like Microsoft® Windows® 95.

A program module may be considered a different version if it uses a different fundamental basis than its previous incarnation. For example, program modules use submodules such as a Dynamic Link Library (DLL) or Application Program Interfaces (APIs). A new program module is a new version if it utilizes a different family of DLLs and APIs than what the previous implementation of such module used.

This same versioning terminology applies to any program module, such as an application, operating system, etc.

Interoperability and Compatibility

Each version of an OS has its corresponding body of applications that are designed to run under it. When a new version of an OS is released, software developers generally upgrade their products to a new version designed to run

under the new OS. Software developers do this for many reasons, including marketing, technology, and economics.

For similar reasons, OS developers wish to make their products backwards compatible. In this way, older versions of applications will run on the latest version of the OS. This encourages users to purchase the new OS because they are not forced to discard their current applications and purchase new versions. This also gives software developers time to provide upgrades to their applications.

Configuration

Configuration is the way a system is set up, or the assortment of components that make up the system. Configuration can refer to either hardware or software, or the combination of both. For instance, a typical configuration for a PC consists of 32MB (megabytes) main memory, a floppy drive, a hard disk, a modem, a CD-ROM drive, a VGA monitor, and an operating system.

Many software products require that the computer have a certain minimum configuration. For example, the software might require a graphics display monitor and a video adapter, a particular microprocessor, and a minimum amount of main memory.

When a person installs a new device or program, she sometimes needs to configure it, which means to set various switches and jumpers (for hardware) and to define values of parameters (for software). For example, the device or program may need to know what type of video adapter you have and what type of printer is connected to the computer. Thanks to new technologies, such as Plug-and-Play, much of this configuration is performed automatically.

Configuration Databases

Software applications typically employ one or more configuration databases to store configuration settings. Under some OSs (such as Windows® 3.1 and MS-DOS®), multiple configuration databases were used by the OS and the applications. There were files for starting the system (e.g., CONFIG.SYS and AUTOEXEC.BAT). There were files for connecting to a network (e.g., NETWORK.INI). There were files for running applications (e.g., WIN.INI and SYSTEM.INI).

Each configuration file had its own rules and structure. Maintaining these files was a difficult chore for the OS. Providing a limited degree of synchronization between these files was also a difficult chore for the OS.

Common Configuration Data Structure

With the advent of more advanced OSs (such as Windows NT® and Windows® 95), a common configuration data structure was introduced. It is called the "Registry." All configuration settings are stored therein (except for other legacy configuration files that remained for backward compatibility reasons).

Herein, a common configuration data structure refers to a set of multiple configuration databases used by more than one version of a program module (such as an application). In addition, a common configuration data structure refers to a single configuration database (such as the Registry) used by more than one version of a program module (such as an application). A configuration database is often stored as one or more configuration files on the storage system of a computer.

1 The Registry. Certain OSs store and check the configuration information
2 (herein, "config-info") at a single location—called the registry. Most applications
3 write data to the registry, at least during installation. The registry is an example of
4 a common configuration data structure.

5 The registry is a hierarchically structured database containing subtrees of
6 keys that contain per-computer and per-user databases. Each key may contain
7 data items called value entries and may contain subkeys. In the registry structure,
8 keys (and subkeys) are analogous to directories and value entries are analogous to
9 files.

10 The registry may include the following major sections (i.e., subtrees):

- 11 • HKEY_Classes_Root - file associations and OLE information
- 12 • HKEY_Current_User - all preferences set for current user
- 13 • HKEY_User - all the current user information for each user of the
14 system
- 15 • HKEY_Local_Machine - settings for hardware, operating system,
16 and installed applications
- 17 • HKEY_Current_Configuration - settings for the display and printers
- 18 • HKEY_Dyn_Data - performance data

19 20 Compatibility Problem

21 Herein, compatibility refers to different versions of the same program
22 module peacefully co-existing on the same system. A new version of a program
23 module is said to be backward compatible if it can use files and data created with
24 an older version of the same program.
25

1 Often different versions of an application store their config-info in a
2 common configuration data structure. In fact, different versions of an application
3 typically store their config-info at the exact same location within a common
4 configuration data structure.

5 A later installed version may overwrite existing config-info for an earlier
6 installed version. As a result, the earlier version is unlikely to run correctly (or at
7 all) because its config-info has been changed. Sometimes residual config-info
8 exists in the common configuration data structure and it may interfere with the
9 smooth performance of the later installed version.

10 By way of example, APIs and DLLs are an area where changes in config-
11 info can greatly affect the performance and/or the operation of different versions
12 of an application using a common configuration data structure. If the later version
13 refers to a family of APIs and/or DLLs that are incompatible with the earlier
14 version, then the earlier version is likely to cause operation failure and/or
15 performance problems. This happens because the earlier version is incapable of
16 using APIs and DLLs designed for use by a later version.

17 18 **Interoperability Problem**

19 As used herein, interoperability is the ability of program modules to share
20 data. In particular, interoperability is the ability of differing "types" of program
21 modules to share data. Program modules are different types when they are not just
22 different versions of each other.

23 An example of interoperability is a Microsoft® Paintbrush application
24 sharing data (such as a bitmap image) with a Microsoft® Office application.
25

1 Regardless of version, Microsoft® Paintbrush and Microsoft® Office are different
2 types of program modules.

3 It is desirable for an OS to provide this type of interoperability between
4 different types of program modules. However, such interoperability is difficult
5 when the program modules are different versions.

6 An example of such a situation is when a 16-bit version of a Microsoft®
7 Paintbrush application wishes to share data (such as a bitmap image) with a 32-bit
8 version of Microsoft® Office application. Not only are these program different
9 types, but also they are different versions.

10 11 **Conventional Solution**

12 The compatibility and interoperability problems like those described above
13 occurred when Microsoft Corporation introduced its 32-bit OS in Windows® 95.
14 To avoid these types of compatibility problems, the new 32-bit applications were
15 instructed to store their config-info in a different location than the older 16-bit
16 applications. In addition, they were directed to use different names for their APIs
17 and DLLs.

18 Although this did help with compatibility and interoperability, it forced
19 software developers to redirect large resources to software conversion rather than
20 to development of new or improved software. Because of these directions,
21 existing 16-bit versions of software needed to be significantly modified to port it
22 to a 32-bit version. The code needed to be altered so that it referred to different
23 APIs and DLLs. The code needed to be altered to store config-info to a different
24 location and access it from that new location.

1
2 **SUMMARY**

3 When using a common configuration data structure (e.g., “registry”), the
4 access redirector and entry reflector promotes compatibility and interoperability
5 between differing versions of program modules.

6 The registry access redirector redirects selected accesses to storage
7 locations (i.e., “loci”, “nodes”) of a common configuration data structure. The
8 selected accesses are redirected to another node. This redirection stores
9 configuration information for differing versions of program modules at different
10 nodes. However, the differing versions believe that they are accessing the same
11 node.

12 As configuration information in a node is changed, the registry entry
13 reflector may copy selected portions of such changed information into its
14 associated “reflected” node and vice versa. This reflection allows associated
15 “reflected” nodes to share relevant configuration information that promotes
16 interoperability.

17
18 **BRIEF DESCRIPTION OF THE DRAWINGS**

19 Fig. 1 illustrates a portion of a typical hierarchical structure of a common
20 configuration data structure that may be a subject of an implementation of
21 an access redirector and entry reflector.

22 Fig. 2 is a flowchart illustrating the methodology of an implementation of
23 an access redirector portion of an access redirector and entry reflector.
24
25

1 Fig. 3 is a flowchart illustrating the methodology of an implementation of
2 an entry reflector portion of an access redirector and entry reflector.

3 Fig. 4 illustrates a portion of a hierarchical structure of a typical "registry",
4 which is an example of a common configuration data structure. The registry may
5 be a subject of an implementation of an access redirector and entry reflector

6 Fig. 5 is an example of a computing operating environment capable of
7 implementing an access redirector and entry reflector.

8 9 **DETAILED DESCRIPTION**

10 The following description sets forth a specific embodiment of an access
11 redirector and entry reflector that incorporates elements recited in the appended
12 claims. This embodiment is described with specificity in order to meet statutory
13 requirements, such as the written description, enablement, and best-mode
14 requirements. However, the description itself is not intended to limit the scope of
15 this patent. Rather, the inventors have contemplated that the claimed invention
16 might also be embodied in other ways, in conjunction with other present or future
17 technologies.

18 19 **Introduction**

20 Described herein is an operating system (OS) that implements the
21 exemplary access redirector and entry reflector ("exemplary redirector/reflector").
22 This version of the OS is designed to accommodate versions of applications
23 designed for the specified version of the OS and versions of applications designed
24 for a previous version of the OS.
25

1 More specifically, described herein is a version of a 64-bit version of an OS
2 (“64-bit OS”) that implements the exemplary access redirector and entry reflector
3 (“exemplary redirector/reflector”). This 64-bit OS is designed to accommodate
4 64-bit versions of applications (64-bit apps) designed for the 64-bit OS and 32-bit
5 versions of applications (32-bit apps) designed for a 32-bit version of an OS (“32-
6 bit OS”).

7 Those who are skilled in the art understand and appreciate that the present
8 invention claimed herein may be implemented in situations other than the specific
9 examples described herein.

10 Registry

11 A “registry” is part of the 64-bit OS, described herein, that implements
12 exemplary redirector/reflector. This “registry” is a common configuration data
13 structure. More particularly, this registry is a common configuration database. In
14 the 64-bit OS described herein, the registry is described in a hierarchical structure.
15 Those who are skilled in the art understand and appreciate that a common
16 configuration data structure (such as the registry described herein) may be
17 organized in any fashion where elements are addressable for storing and accessing.
18

19 Fig. 1 illustrates a portion of the hierarchical structure of the exemplary
20 registry at 100. The top of the registry is root 102 called “registry tree.” Below
21 that are several subtrees. The head of two of such subtrees are shown in Fig. 1:
22 “SubTreeA” 112 and “SubTreeB” 110. None of the nodes of SubTreeA are shown
23 in Fig. 1. However, some of the nodes of SubTreeB are shown in this illustration.
24
25

1 “Branch” 120 and “SubBranch” 150 are examples of successively deeper
2 nodes in the SubTreeB 110. Under Branch 120 is “Table” node 130. Under node
3 130 is a table 132. In the table is a list of addresses (which are discussed below)
4 used for the implementation of the exemplary redirector/reflector. These are
5 addresses to nodes or loci in the registry. Thus, the table 132 may be called a
6 “loci-redirection” or “node-remapping” table.

7 Under SubBranch 150 are multiple nodes, such as “Twig” 160 shown in
8 Fig. 1. Under Twig 160 are one or more leafs, such as “Leaf” 162. Twig 160 and
9 its leaf 162 are on the “default side” of line 180. On the other side of the line are
10 “Proxy Twig” 170 and “Mirror Leaf” 172. This other side of line 180 is called the
11 “mirror” side.

12 Each node in the exemplary registry is addressable using this generic
13 notation:

14
15 “RegistryTree/SubTreeX/Branch/SubBranch/Twig/Leaf”
16

17 This example refers to the Leaf 162 node. Of course, any addressing scheme may
18 be used. This one is used to facilitate description of the exemplary
19 redirector/reflector.

20 21 **Redirector**

22 The exemplary redirector/reflector includes a “redirector” portion. The
23 exemplary redirector promotes compatibility.
24
25

The exemplary redirector intercepts registry accesses and forwards them to a new location within the registry. The effect of such redirection is that registry reads and writes may be at a different location within the registry than what read and write commands specified.

In other words, the exemplary redirector will sit and wait for an application to access the registry. When it does, the application provides an address of a node where it is attempting to write or read. The access is intercepted. The type of access may be examined. Also, the application attempting to access is examined.

Based upon such examination and based upon a table of remapped addresses, the exemplary redirector may redirect such access to another node. Otherwise, the exemplary redirector may allow such access to the specified node. In either situation, the application believes that it is accessing the specified node.

The remapped nodes are for differing versions of application. For example, the default node may be for a 64-bit application on a 64-bit OS and the remapped node may be a 32-bit version of the same application.

Problems are inevitable when multiple independent entities write to and read from the same storage location in a database and when those independent entities believe that they have sole control over the information stored there. Here, each version of the same application believes that it has exclusive control over information stored at the same particular node in the registry. Therefore, there is a strong possibility that information added, modified, and/or removed by one application will significantly affect the other. The solution provided by the exemplary redirector is to redirect access to a unique location for each version of an application.

Redirector Methodology Implementation

Fig. 2 shows a methodological implementation of the exemplary redirector. At 200, an application attempts to access a particular node in the registry. Access typically includes reading and/or writing.

Assume for descriptive purposes that the application uses this particular requested node to store config-info about itself. This node may specify associated DLLs and APIs. It may specify initialization and customization options.

At 202, the exemplary redirector intercepts such access and examines it. It may examine any aspect of the access: the type of access, the timing of such access, the parameters, etc. It may also examine the application requesting such access. At 204, based upon such examination, the exemplary redirector will determine whether such access should be redirected. If no redirection is necessary, then the process jumps to block 210 where access is allowed to the requested node. Otherwise, the process proceeds to block 206.

At 206, the exemplary redirector examines a look-up table of remapped addresses. If the address of requested node is not found in the table, then the original address of the node is specified at 208. If the address of requested node is found in the table, then its associated remapped address is specified at 208. At 210, this specified address is used to write to the registry or to read from the registry.

Example using exemplary redirector methodological implementation of Fig. 2. The explanation of this example refers to Figs. 1 and 2. The registry is a common configuration data structure for a 64-bit OS and applications running

1 thereunder. At 200 in Fig. 2, a 32-bit version application attempts to access a
2 particular node in the registry. The particular node in this example is
3 “RegistryTree/SubTreeX/Branch/SubBranch/Twig/Leaf” (i.e., node 162 in Fig. 1).

4 Assume for descriptive purposes that the 32-bit application uses this
5 particular requested node 162 to store config-info about itself. Also, assume that a
6 64-bit version of the same application uses this same node 162 to store config-info
7 about itself.

8 At 202, the exemplary redirector intercepts the 32-bit application’s access
9 to “RegistryTree/.../Twig/Leaf” and examines it. It determines the version of the
10 application requesting such access. At 204, since a 32-bit application is requesting
11 such access, it determines that it should be redirected. Thus, the process proceeds
12 to block 206.

13 At 206, the exemplary redirector examines the look-up table (132 of Fig. 1)
14 of remapped addresses. It finds “RegistryTree/.../Twig/Leaf” in the table 132 and
15 knows that such address should be redirected to its associated remapped address in
16 the table: “RegistryTree/.../ProxyTwig/MirrorLeaf.” At 208, the remapped
17 address is specified. At 210, this specified address is used to write to the registry
18 or to read from the registry at “RegistryTree/.../ProxyTwig/MirrorLeaf” rather
19 than the originally requested address. The remapped node is Mirror Node 172.

20 Thus, each time that the 32-bit application reads or writes anything to
21 “RegistryTree/.../Twig/Leaf” node, it instead reads or writes to
22 “RegistryTree/.../ProxyTwig/MirrorLeaf” node.
23
24
25

Reflector

The exemplary redirector/reflector includes a “reflector” portion. The exemplary reflector promotes interoperability.

As needed, the exemplary reflector examines mirrored nodes. Differences are noted. Those differences in one node that promote interoperability are copied to the other node. Thus, some modifications of a node are “reflected” in its associated mirror node and vice versa.

Reflector Methodology Implementation

Fig. 3 shows a methodological implementation of the exemplary reflector. At 300, the exemplary reflector waits for a triggering event. One example of such an event is a notification from the OS that a change has been made to the registry. Another example may be a passage of a given amount of time.

At 302, the exemplary reflector reads a look-up table of reflected nodes. At 304, it examines entries of each reflected node in the table. It determines if any changes were made to any of the nodes (on both the default and the mirror sides).

At 306, the exemplary reflector determines whether mirroring such changes will promote interoperability. If so, then it mirrors the changes in one node to its associated node. In other words, the changes in the reflected node are stored in the original node and changes in the original node are stored in the reflected node.

Example using exemplary reflector methodological implementation of Fig.

3. The explanation of this example refers to Figs. 1 and 3. The registry is a

common configuration data structure for a 64-bit OS and applications running thereunder.

At 300 of Fig. 3, the OS notifies the exemplary reflector that there has been a change to the registry. For this example, the change will be to node 172 of Fig. 1 in accordance to the above description of the example using exemplary redirector methodological implementation of Fig. 2.

At 302 of Fig. 3, the exemplary reflector_reads a look-up table of reflected nodes. At 304, it examines entries of each reflected node in the table. During this examination, it discovers the node 172 of Fig. 1 has been altered.

At 306 of Fig. 3, the exemplary reflector determines whether mirroring such changes will promote interoperability. For this example, assume that it does promote interoperability. So, the exemplary reflector_mirrors the changes in node 172 to its associated node 162.

Redirector and Reflector

In the exemplary redirector/reflector, the above-described exemplary redirector and exemplary reflector are implemented concurrently. The combination promotes compatibility and interoperability.

Redirector/Reflector Implementation Example

The following is an example illustrating how the exemplary redirector/reflector promotes interoperability by redirecting/reflecting as appropriate:

32-bit application writes a new key under

- \\Registry\\Machine\\Software\\Classes\\Clsid {0285b5c0-12c7-11ce-bd31-00aa004bbb1f}\\LocalServer32 REG_SZ "c:\\Office\\winword.exe";

- the registry redirector intercepts this attempt to create the key and redirects to

```
\\Registry\\Machine\\Software\\Wow6432Node\\Classes\\Clsid\\  
{0285b5c0-12c7-11ce-bd31-0aa004bbb1f}\\LocalServer32;  
REG SZ "c:\\Office\\winword.exe";
```

- the reflector then sees the key created, and sees that `\\Registry\\Machine\\Software\\Wow6432Node\\Classes` is in the list of keys that may potentially be reflected;
- the reflector examines the newly created key, and sees that it is a `LocalServer32` key. The reflector contains hard-coded knowledge that `LocalServer32` keys should be reflected, as they indicate that the GUID (unique program identifier) corresponds to an EXE file. It knows that 32-bit and 64-bit OLE/COM applications can interop with opposite-architecture OLE/COM EXEs, so the key should be reflected;
- the reflector then copies the contents of `\\Registry\\Machine\\Software\\Wow6432Node\\Classes\\Clsid\\{0285b5c0-12c7-11ce-bd31-00aa004bbb1f}\\LocalServer32` to `\\Registry\\Machine\\Software\\Classes\\Clsid\\{0285b5c0-12c7-11ce-bd31-00aa004bbb1f}\\LocalServer32`

1 The following is a counter-example illustrating when data is not reflected
2 using the exemplary reflector:

- 3 • imagine a 32-bit app creating a new key under
4 \\Registry\\Machine\\Software\\Classes\\Clsid that looks like:
5 {0285b5c0-12c7-11ce-bd31-00aa004bbb1f}\\InprocServer32
6 REG_SZ "c:\\Office\\spwordbrk.dll"
- 7 • This one will be redirected to
8 \\Registry\\Machine\\Software\\Wow6432Node\\Classes\\Clsid just
9 like the above example, but the redirector will look at
10 "InprocServer32" and know that the OLE/COM interface is
11 implemented in a DLL. A 32-bit DLL cannot be loaded into a 64-bit
12 process due to the limitations of the WOW64 emulator, so the
13 reflector will not reflect this key over to the 64-bit registry view.

14 **Implementation Details of an Exemplary Embodiment**

15 For this example, assume that an exemplary 64-bit OS is designed to
16 support two versions of applications, 32-bit and 64-bit. To maintain configuration
17 information ("config-info"), this OS maintains a common configuration data
18 structure, called the registry. Fig. 4 shows an example of a portion of such a
19 registry. Both the 32-bit and 64-bit apps share config-info stored in the registry
20 without affecting each other.

21 For example, a 64-bit app might create a key in the registry for its own
22 purpose (like "ClassID") and a 32-bit app might not be familiar with that new key,
23 but it should not delete it. Some applications might need to load different
24
25

1 components using some well-known GUID information. (A GUID is a unique
2 code that identifies an interface to an object across all computers and networks.)

3 Fig. 4 illustrates a portion of the hierarchical structure of the exemplary
4 registry at 400. The top of the registry is root 402 called "registry." Below that
5 are several subtrees. The head of two of such subtrees are shown in Fig. 4:
6 "Machine" 412 and "User" 410. None of the nodes of Machine is shown in Fig. 4.
7 However, some of the nodes of User are shown in this illustration.

8 "Software" 420 and "Class" 450 are examples of successively deeper nodes
9 in the User 410. Under Software 420 is "ISN Master Table" node 430. ISN is
10 Image Specific Node. As used here, "image" refers to a representation of a
11 collection of data stored in a section of memory. More specifically, the term
12 "image specific" refers to a representation of a specific application in memory.
13 Under node 430 is the master ISN table 432.

14 Under Class 450 are multiple ISN nodes, such as CLaS Identifier
15 (CLSID) 460 shown in Fig. 4. Under CLSID 460 are one or more ISN identifier
16 nodes, such as "GUID1" 462. CLSID 460 and its child 462 are on the "default
17 side" of line 480. On the other side of the line are "Wow6432Node" 470 and
18 "GUID1" 472. This other side is called the "mirror" side.

19 The exemplary redirector is responsible for policing registry-access calls
20 initiated by a 32-bit or 64-bit application. The apps will use the same registry, but
21 the exemplary redirector re-interprets registry information as appropriate for the
22 application. To promote interoperability, it helps 32-bit applications to share some
23 config-info with the 64-bit applications.
24
25

1 Image Specific Node (ISN): There might be some ISN in the tree that will
2 be visible only to a particular type of image. For example, some key might be
3 visible only to a 64-bit image, whereas some key might be visible only to a 32-bit
4 image. The location of a 64-bit component stored in the registry might not be
5 relevant to 32-bit applications whereas a 32-bit version of the same component
6 might be relevant to a 32-bit application.

7
8 Registry organization:

9 1. A master table (such as 432 of Fig. 4) in the registry defining ISN in
10 the registry. ISN node will specify that all the children beyond ISN node will be
11 specific to a particular type of image (i.e., either 32-bit or 64-bit).

12 2. The exemplary redirector splits the tree at ISN node (such as node
13 460) putting "Wow6432Node" node 470 as a child.

14 3. Any call from 32-bit applications beyond ISN node (such as node
15 460) will be redirected to some child under Wow6432Node 470.

16 4. Redirection of calls from 64-bit or 32-bit applications are
17 transparent. For example, 32-bit applications will access the registry using the
18 same key name it used in the true 32-bit world.

19 5. Redirector will use "Wow6432Node" 470 as a point of redirection.

20 6. A setup program will copy a necessary key from the 64-bit tree to
21 the 32-bit tree so that 32-bit applications can work properly.

22 7. The exemplary reflector (as a background thread) runs occasionally
23 to copy back and forth information that needs to be shared on the both sides of the
24 mirror line 480.
25

1
2 Master ISN Table

3 A master ISN table (such as 432) might have the following two entries for
4 example:

- 5
6 1. \\REGISTRY\\USER*\\Software\\Classes\\CLSID
7 2. \\REGISTRY\\MACHINE\\SOFTWARE\\Microsoft\\Shared Tools
8

9 In this example, the CLSID 460 has one key GUID1 462. The setup
10 program creates another key "Wow6432Node" 470 under CLSID for the
11 redirection and copy GUID1 472 under Wow6432Node for use by 32-bit
12 applications.
13

14 Scenarios:

15 1. 32-bit applications try to open some CLSID (such as GUID1 462)
16 under \\REGISTRY\\USER*\\Software\\Classes\\CLSID. Applications can open
17 that key in multiple calls or a single call using RegOpenKey. Eventually, they will
18 end up opening the key under \\REGISTRY\\USER*\\Software
19 \\Classes\\CLSID\\Wow6432Node 470. Client can use the handle as if they have
20 opened the right key because the redirection is completely transparent to the
21 applications.
22

23 2. 64-bit applications try to open some CLSID (such as GUID1 462)
24 under \\REGISTRY\\USER*\\Software\\Classes\\CLSID. The call will not be
25 redirected and the 64-bit application will get a handle to the right key.

1 3. 32-bit applications pass a handle to 64-bit applications. If the handle
2 has already been redirected, then any call by the 64-bit application will go to the
3 child under the redirected sub-tree.

4 4. 64-bit applications pass a handle to 32-bit applications. By default,
5 64-bit applications cannot get a redirected handle. 32-bit applications will not
6 redirect the path up to the key pointed by the handle. In other words, in the above
7 example if 64-bit applications pass a handle to GUID1 to 32-bit applications, then
8 any open call by 32-bit applications using that handle will not be redirected. But if
9 the 64-bit applications pass the handle to
10 \\REGISTRY\\USER*\\Software\\Classes\\CLSID, then any subsequent call by
11 32-bit applications will be redirected.

12 13 **Exemplary Alternative Implementations**

14 While the above examples are described in terms of a default side and a
15 mirrored side, an alternative implementation may have no default side. An
16 alternative exemplary redirector/reflector may simply make a determination each
17 time and no side is favored.

18 Moreover, there may be more than two "sides". The registry may have
19 multiple reflected nodes. Each node represents a specific image (such as an
20 application).

21 Although the above examples are described in terms of versions where the
22 differences is based upon the fundamental design of the OS, a version of a
23 program module may differ from another when each stores config-info in a
24 common location.

1 While the above examples are described in terms of applications, the
2 exemplary redirector/reflector may be any program module or sub-module. That
3 includes, but is not limited to, OSs, applications, APIs, DLLs, program objects,
4 and procedures.

5 Although the above examples are described in terms of a 32-bit OS and a
6 64-bit OS, the exemplary redirector/reflector may be implemented on any
7 generation of OS. That includes, but is not limited to, 32-bit OS, 64-bit OS, 128-
8 bit OS, and 256-bit OS.

9 The above examples discuss differing OSs, but the exemplary
10 redirector/reflector may be implemented with applications designed for the same
11 version of OS, but may use different elements (such as DLLs, APIs, data files and
12 formats, etc.).

13 **Exemplary Computing Environment**

14
15 Fig. 5 illustrates an example of a suitable computing environment 920 on
16 which the exemplary redirector/reflector may be implemented.

17 Exemplary computing environment 920 is only one example of a suitable
18 computing environment and is not intended to suggest any limitation as to the
19 scope of use or functionality of the exemplary redirector/reflector. Neither should
20 the computing environment 920 be interpreted as having any dependency or
21 requirement relating to any one or combination of components illustrated in the
22 exemplary computing environment 920.

23 The exemplary redirector/reflector is operational with numerous other
24 general purpose or special purpose computing system environments or
25

1 configurations. Examples of well known computing systems, environments,
2 and/or configurations that may be suitable for use with the exemplary
3 redirector/reflector include, but are not limited to, personal computers, server
4 computers, thin clients, thick clients, hand-held or laptop devices, multiprocessor
5 systems, microprocessor-based systems, set top boxes, programmable consumer
6 electronics, network PCs, minicomputers, mainframe computers, distributed
7 computing environments that include any of the above systems or devices, and the
8 like.

9 The exemplary redirector/reflector may be described in the general context
10 of computer-executable instructions, such as program modules, being executed by
11 a computer. Generally, program modules include routines, programs, objects,
12 components, data structures, etc. that perform particular tasks or implement
13 particular abstract data types. The exemplary redirector/reflector may also be
14 practiced in distributed computing environments where tasks are performed by
15 remote processing devices that are linked through a communications network. In
16 a distributed computing environment, program modules may be located in both
17 local and remote computer storage media including memory storage devices.

18 As shown in Fig. 5, the computing environment 920 includes a general-
19 purpose computing device in the form of a computer 930. The components of
20 computer 920 may include, by are not limited to, one or more processors or
21 processing units 932, a system memory 934, and a bus 936 that couples various
22 system components including the system memory 934 to the processor 932.

23 Bus 936 represents one or more of any of several types of bus structures,
24 including a memory bus or memory controller, a peripheral bus, an accelerated
25

1 graphics port, and a processor or local bus using any of a variety of bus
2 architectures. By way of example, and not limitation, such architectures include
3 Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA)
4 bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA)
5 local bus, and Peripheral Component Interconnects (PCI) bus also known as
6 Mezzanine bus.

7 Computer 930 typically includes a variety of computer readable media.
8 Such media may be any available media that is accessible by computer 930, and it
9 includes both volatile and non-volatile media, removable and non-removable
10 media.

11 In Fig. 5, the system memory includes computer readable media in the form
12 of volatile memory, such as random access memory (RAM) 940, and/or non-
13 volatile memory, such as read only memory (ROM) 938. A basic input/output
14 system (BIOS) 942, containing the basic routines that help to transfer information
15 between elements within computer 930, such as during start-up, is stored in ROM
16 938. RAM 940 typically contains data and/or program modules that are
17 immediately accessible to and/or presently be operated on by processor 932.

18 Computer 930 may further include other removable/non-removable,
19 volatile/non-volatile computer storage media. By way of example only, Fig. 5
20 illustrates a hard disk drive 944 for reading from and writing to a non-removable,
21 non-volatile magnetic media (not shown and typically called a "hard drive"), a
22 magnetic disk drive 946 for reading from and writing to a removable, non-volatile
23 magnetic disk 948 (e.g., a "floppy disk"), and an optical disk drive 950 for reading
24 from or writing to a removable, non-volatile optical disk 952 such as a CD-ROM,
25

1 DVD-ROM or other optical media. The hard disk drive 944, magnetic disk drive
2 946, and optical disk drive 950 are each connected to bus 936 by one or more
3 interfaces 954.

4 The drives and their associated computer-readable media provide
5 nonvolatile storage of computer readable instructions, data structures, program
6 modules, and other data for computer 930. Although the exemplary environment
7 described herein employs a hard disk, a removable magnetic disk 948 and a
8 removable optical disk 952, it should be appreciated by those skilled in the art that
9 other types of computer readable media which can store data that is accessible by a
10 computer, such as magnetic cassettes, flash memory cards, digital video disks,
11 random access memories (RAMs), read only memories (ROM), and the like, may
12 also be used in the exemplary operating environment.

13 A number of program modules may be stored on the hard disk, magnetic
14 disk 948, optical disk 952, ROM 938, or RAM 940, including, by way of example,
15 and not limitation, an operating system 958, one or more application programs
16 960, other program modules 962, and program data 964. Each of such operating
17 system 958, one or more application programs 960, other program modules 962,
18 and program data 964 (or some combination thereof) may include an embodiment
19 of the exemplary redirector/reflector. More specifically, each may include an
20 embodiment of an access-redirector, an entry-reflector, a receiver, a searcher, a
21 loci-access redirector, a common configuration data structure, director, and
22 replicator.

23 A user may enter commands and information into computer 930 through
24 input devices such as keyboard 966 and pointing device 968 (such as a "mouse").
25

1 Other input devices (not shown) may include a microphone, joystick, game pad,
2 satellite dish, serial port, scanner, or the like. These and other input devices are
3 connected to the processing unit 932 through a user input interface 970 that is
4 coupled to bus 936, but may be connected by other interface and bus structures,
5 such as a parallel port, game port, or a universal serial bus (USB).

6 A monitor 972 or other type of display device is also connected to bus 936
7 via an interface, such as a video adapter 974. In addition to the monitor, personal
8 computers typically include other peripheral output devices (not shown), such as
9 speakers and printers, which may be connected through output peripheral interface
10 975.

11 Computer 930 may operate in a networked environment using logical
12 connections to one or more remote computers, such as a remote computer 982.
13 Remote computer 982 may include many or all of the elements and features
14 described herein relative to computer 930.

15 Logical connections shown in Fig. 5 are a local area network (LAN) 977
16 and a general wide area network (WAN) 979. Such networking environments are
17 commonplace in offices, enterprise-wide computer networks, intranets, and the
18 Internet.

19 When used in a LAN networking environment, the computer 930 is
20 connected to LAN 977 via network interface or adapter 986. When used in a
21 WAN networking environment, the computer typically includes a modem 978 or
22 other means for establishing communications over the WAN 979. The modem
23 978, which may be internal or external, may be connected to the system bus 936
24 via the user input interface 970, or other appropriate mechanism.
25

1 Depicted in Fig. 5, is a specific implementation of a WAN via the Internet.
2 Computer 930 typically includes a modem 978 or other means for establishing
3 communications over the Internet 980. Modem 978, which may be internal or
4 external, is connected to bus 936 via interface 970.

5 In a networked environment, program modules depicted relative to the
6 personal computer 930, or portions thereof, may be stored in a remote memory
7 storage device. By way of example, and not limitation, Fig. 5 illustrates remote
8 application programs 989 as residing on a memory device of remote computer
9 982. It will be appreciated that the network connections shown and described are
10 exemplary and other means of establishing a communications link between the
11 computers may be used.
12

13 **Exemplary Operating Environment**

14 Fig. 5 illustrates an example of a suitable operating environment 920 in
15 which the exemplary redirector/reflector may be implemented. Specifically, the
16 exemplary redirector/reflector is implemented by any program 960-962 or
17 operating system 958 in Fig. 5.

18 The operating environment is only an example of a suitable operating
19 environment and is not intended to suggest any limitation as to the scope or use of
20 functionality of the exemplary redirector/reflector described herein. Other well
21 known computing systems, environments, and/or configurations that may be
22 suitable for use with the exemplary redirector/reflector include, but are not limited
23 to, personal computers (PCs), server computers, hand-held or laptop devices,
24 multiprocessor systems, microprocessor-based systems, programmable consumer
25

1 electronics, wireless phones and equipments, general- and special-purpose
2 appliances, network PCs, minicomputers, mainframe computers, distributed
3 computing environments that include any of the above systems or devices, and the
4 like.

5 6 **Computer-Executable Instructions**

7 An implementation of the exemplary redirector/reflector may be described
8 in the general context of computer-executable instructions, such as program
9 modules, executed by one or more computers or other devices. Generally,
10 program modules include routines, programs, objects, components, data structures,
11 etc. that perform particular tasks or implement particular abstract data types.
12 Typically, the functionality of the program modules may be combined or
13 distributed as desired in various embodiments.

14 15 **Computer Readable Media**

16 An implementation of the exemplary redirector/reflector may be stored on
17 or transmitted across some form of computer readable media. Computer readable
18 media can be any available media that can be accessed by a computer. By way of
19 example, and not limitation, computer readable media may comprise computer
20 storage media and communications media.

21 Computer storage media include volatile and non-volatile, removable and
22 non-removable media implemented in any method or technology for storage of
23 information such as computer readable instructions, data structures, program
24 modules, or other data. Computer storage media includes, but is not limited to,
25

1 RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM,
2 digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic
3 tape, magnetic disk storage or other magnetic storage devices, or any other
4 medium which can be used to store the desired information and which can be
5 accessed by a computer.

6 Communication media typically embodies computer readable instructions,
7 data structures, program modules, or other data in a modulated data signal such as
8 carrier wave or other transport mechanism and included any information delivery
9 media. The term "modulated data signal" means a signal that has one or more of
10 its characteristics set or changed in such a manner as to encode information in the
11 signal. By way of example, and not limitation, communication media includes
12 wired media such as a wired network or direct-wired connection, and wireless
13 media such as acoustic, RF, infrared, and other wireless media. Combinations of
14 any of the above are also included within the scope of computer readable media.
15

16 Conclusion

17 Although the access redirector and entry reflector has been described in
18 language specific to structural features and/or methodological steps, it is to be
19 understood that an access redirector and entry reflector defined in the appended
20 claims is not necessarily limited to the specific features or steps described. Rather,
21 the specific features and steps are disclosed as preferred forms of implementing
22 the claimed invention.
23
24
25